CHAPTER 3

SOFTWARE

LEARNING OBJECTIVES

Upon completion of this chapter, you will be able to do the following:

1. Recognize and compare the different types and functions of operating systems.
2. Identify the types of utilities and explain their functions.
3. Describe the different types and functions of programming languages.
4. Explain the steps necessary to develop a program and describe the tools used.
5. Compare and describe the types and functions of applications packages.

INTRODUCTION

Up to now we have been discussing computer OPERATIONAL CONCEPTS and HARDWARE (the computer and its peripheral devices), and how these devices work and communicate with each other. What about this thing called SOFTWARE? Do we really need it? We most certainly do! Software plays a major role in computer data processing. For example, without software, the computer could not perform simple addition. It's the software that makes everything happen. Or putting it another way, software brings the computer to life.

You already know it takes a program to make the computer function. You load an operating system into the computer to manage the computer's resources and operations. You give job information to the operating system to tell it what you want the computer to do. You may tell it to assemble or compile a COBOL program. You may tell it to run the payroll or print inventory reports. You may tell it to copy a tape using a utility program. You may tell it to print the data from a disk file, also using a utility program. You may tell it to test a program. This job information may be entered through the console or read into the computer from tape or disk. It also may be entered by the programmer or user from a remote computer terminal. The operating system receives and processes the job information and executes the programs according to that job information.

Software can be defined as all the stored programs and routines (operating aids) needed to fully use the capabilities of a computer. Generally speaking, we say, "If it is not hardware then it must be software."

OPERATING SYSTEMS

The operating system is the heart of any computer system. Through it, everything else is done. Basically, operating systems are designed to provide the operator with the most efficient way of executing many user programs.

An operating system is a collection of many programs used by the computer to manage its own resources and operations. These programs control the execution of other programs. They schedule, assign resources, monitor, and control the work of the computer. There are several types.

TYPES OF OPERATING SYSTEMS

Operating systems are designed to provide various operating modes. Some systems can only do one task at a time, while others can perform several at a time. Some systems allow only one person to use the system, and others allow multiple users. Single user/single tasking operating systems are the simplest and most common on microcomputers. CP/M®-80[1], CP/M-86®[1], and MS-DOS®[2] are examples. Single user/multitasking operating systems allow you to do more than one task as long as the tasks don't use the same type of resources. For example, you can print one job while you run another, as long as the second job does not require the printer. Examples are Concurrent CP/M® -86 [3], Concurrent® DOS [3], and MS-DOS®; (3.0 and above). Multiuser/multitasking operating systems let more than one user access the same resources at the same time. This is especially useful for sharing common data. These are only feasible on processors (the functional unit in a computer that interprets and executes instructions) of 16 bits or more and with large memories. UNIX [4] is an example. There are also multiprocessor systems, shared resource systems. This means each user (or operator) has a dedicated microprocessor (cpu), which shares common resources (disks, printers, etc.).

1. CP/M and CP/M-86 are registered trademarks of Digital Research Inc.

2. MS-DOS is a registered trademark of Microsoft Corporation.

3. Concurrent CP/M and Concurrent DOS are trademarks of Digital Research Inc.

4. UNIX is a trademark of AT&T.

COMPATIBILITY WITH APPLICATIONS SOFTWARE

To use an applications program, it must be compatible with the operating system. Therefore, the availability of application software for a particular operating system is critical. Because of this, several operating systems have become the most popular. For 8-bit microcomputers, CP/M (Control Program for Microprocessors) is widely used because many hardware manufacturers have adopted it. MS-DOS (MicroSoft Disk Operating System) designed from CP/M dominates in lower performance 16-bit systems. UNIX, an operating system for larger computers, is being used on the more powerful 16-bit and 32-bit microcomputers. Other operating systems are offered by microcomputer manufacturers.

To overcome the applications software compatibility problem, some software comes in several versions so it can be run under several different operating systems. The point to remember is that not all applications software will run on all systems. You have to check to see that compatibility exists. You need the right version.

OPERATING SYSTEM FUNCTIONS

To give you a better idea of what you can expect to see on your microcomputer display screen, we will show a few fundamental disk operating system commands and messages. Again, the functions of each operating system are about the same, but each may use a different command to do about the same thing. For example, try not to get confused because CP/M uses the command PIP (peripheral interchange program) to copy a file, while MS-DOS uses the command COPY.

Remember, the first thing you need to do is boot (initial program load) the system. There are many ways this can be done. Here is an example. When you turn on the power, a prompt may appear on the screen. You then insert the operating system floppy disk into the drive A. Type a B (for boot) and press the RETURN key. The operating system will load from the disk. If you are using a system set up for automatic booting, you won't have to type the B. The system automatically loads the operating system when you insert the disk that contains it. Some systems will then ask for date and time. Enter these. You will next see a prompt, usually A> (or A:). The system is ready and drive A is assigned as your primary drive. One thing you might want to do is to display the disk directory to see what is on the disk. To do this, enter DIR following the A>. This will list your files.

COMMAND	.COM
CONFIGUR	.COM
DATDBL	.BAK
DATDBL	.DOC
FINANCE	.BAS
MASTER	.DOC

It may also give you file size and the date and time of the file. Let's take an example. Let's say you are to copy the file "MASTER.DOC" from the floppy disk in drive A to the floppy disk in drive B and then delete the file on the floppy disk in drive A. You have just displayed the directory of the floppy disk in drive A. Check to see that the file you want is on the floppy disk in drive A. It is. You then insert the floppy disk on which you want the copy into drive B. Be sure it is formatted with the track and sector information so it is ready to receive data. Also, be sure the disk is not write-protected. On a 5-1/4 inch floppy disk that means the write protect notch is uncovered. Following the A> type

COPY MASTER.DOC B:

and press RETURN. The system will copy the file and give it the same name. Next you might want to display the directory on drive B to see that the file was copied. You can do this by entering DIR B: following the A> prompt. To delete the file on the floppy disk in drive A, type

DEL MASTER.DOC

following the A> prompt on the screen and press RETURN.

You probably noticed each entry in the directory is followed by three characters. These are called extensions, and we use them to tell us the type of file we are working with. For example,

.BAK	Means backup file.
.BAS	Means BASIC source program.
.TMP	Means temporary file.
.DOC	Means ASCII document file.
.BIN	Means binary file, and so on.

Other typical built-in operating system commands you can use might include:

RENAME	to change the name of a file
DISKCOPY	to copy a whole floppy disk
FORMAT	to initialize a floppy disk, get it ready to receive data and programs from the system
TIME	to display or set the time

You will learn to use these and many other system commands as you operate a specific computer.

We won't go into any more detail here. You will have documentation and reference manuals for the specific version of the operating system you will be using.

Q-1. What is the heart of any computer system?

Q-2. Which types of operating systems are the simplest and most common on microcomputers?

Q-3. What types of operating systems let more than one user access the same resources at the same time?

Q-4. Why is the availability of applications software for a particular operating system critical?

Q-5. How is the applications software compatibility problem overcome?

UTILITY PROGRAMS

Now that you have learned about operating systems, let's go into another type of program, utilities. In addition to the utility commands (like diskcopy and rename), which are built into the operating system, you will probably have some independent utility programs. These are standard programs that run under control of the operating system just like your applications programs. They are called utilities because they perform general types of functions that have little relationship to the content of the data. Utility programs eliminate the need for programmers to write new programs when all they want to do is copy, print, or sort a data file. Although a new program is not needed, we do have to tell the program what we want it to do. We do this by providing information about files, data fields, and the process to be used. For example, a sort program arranges data records in a specified order. You will have to tell the sort program what fields to sort on and whether to sort in ascending or descending sequence.

Let's examine two types of utility programs to give you some idea of how a utility program works. The first will be sort-merge and the second the report program generator (RPG).

SORT-MERGE PROGRAMS

Sorting is the term given to arranging data records in a predefined sequence or order. Merging is the combining of two or more ordered files into one file. For example, we normally think of putting a list of people's names in alphabetical order arranging them in sequence by last name. We arrange those with the same last name in order by first name.

If we do this ourselves, we know the alphabetic sequence —B comes after A, C after B, and so on, and it is easy to arrange the list, even if it is a time consuming job. On a computer, the sequence of characters is also defined. It is called the collating sequence. Every coding system has a collating sequence. The capability of a computer to compare two values and determine which is greater (B is greater than A, C is greater than B, and so on) makes sorting possible. What about numbers and special characters? They are also part of the collating sequence. In EBCDIC, (EBCDIC is a computer code that is discussed in detail in chapter 4) special characters, such as #, $, &, and *, come in front of the alphabetic characters, and numbers follow. When you sort records in the defined sequence, they are in ascending sequence. Most sort programs also allow you to sort in reverse order. This is called descending sequence. In EBCDIC, it is 9-0, Z-A, then special characters.

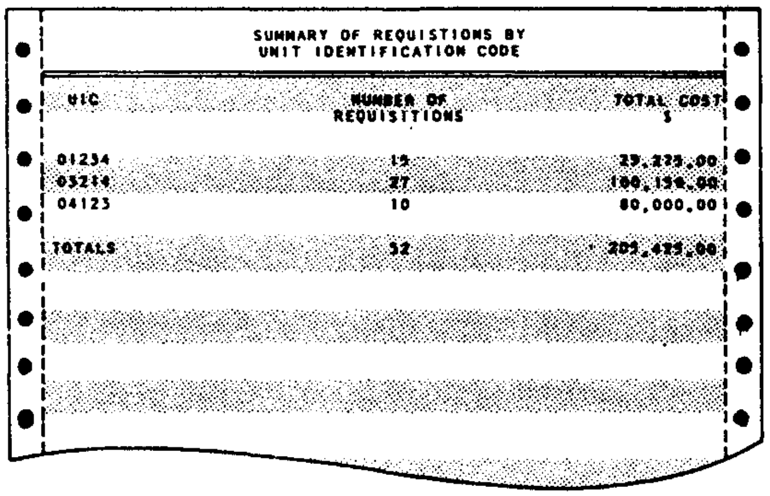
To sort a data file, you must tell the sort program what data field or fields to sort on. These fields are called sort (or sorting) keys. In our example, the last name is the major sort key and the first name is the minor sort key.

Sorting is needed in many applications. For example, for mailing we need addresses in ZIP Code order; personnel records may be kept in service number order; and inventory records may be kept in stock number order. We could go on and on. Because many of our files are large, sorting is very time consuming, and it is one of the processes most used on computer systems. As a user, you will become very familiar with this process.

Sort-merge programs usually have phases. First they initialize: read the parameters, produce the program code for the sort, allocate the memory space, and set up other functions. The sort-merge program then reads in as many input data records as the memory space allocated can hold, arranges (sorts) them in sequence, and writes them out to an intermediate sort-work file. It continues reading input, sorting and writing intermediate sort-work files until all the input is processed. It then merges (combines) the ordered intermediate sort-work files to produce one output file in the sequence specified. The merging process can be accomplished with less memory than the sort process since the intermediate sort work files are all in the same sequence. Records from each work file can be read, the sort keys compared based on the collating sequence and sort parameters, and records written to the output file maintaining the specified sequence.

REPORT PROGRAM GENERATORS

Report program generators (RPG) are used to generate programs to print detail and summary reports of data files. Figure 3-1 is an example of a printed report. RPGs were designed to save programming time. Rather than writing procedural steps in a language like BASIC or COBOL, the RPG programmer writes the printed report requirements on specially designed forms.



```
                 SUMMARY OF REQUISTIONS BY
                 UNIT IDENTIFICATION CODE

 UIC                 NUMBER OF              TOTAL COST
                    REQUISITIONS                 $

 01234                   15                  25,275.00
 03214                   27                 100,150.00
 04123                   10                  80,000.00

TOTALS                   52                 205,425.00
```

Figure 3-1.—Printed report using a report program generator (RPG) program.

Included in the requirements are an input file description, the report heading information lines, the input data record fields, the calculations to be done, and the data fields to be printed and summarized. The RPG program takes this information and generates a program for the specific problem. You then run that program with the specified input data file to produce the printed report. The input data file must be in the sequence in which you want the report to summarize the data.

In our example (fig. 3-1), we summarized requisitions based on unit identification codes (UIC). We first sorted the input data file on the field that contains the UIC. We provided specifications to the RPG program to tell it to accumulate totals from the detail (individual) data records until the UIC changed. We then told it to print the total number of requisitions and total cost for that UIC. We did not have it print each detail record, although we could have. The UIC is called the control field. Each time the control field changes, there is a control break. Each time there is a control break, the program prints the summary information. After all records are read and processed, it prints a summary line (TOTALS) for all UICs. You can also use RPGs to generate a program to update data files.

Q-6. What programs eliminate the need for programmers to write new programs when all they want to do is copy, print, or sort a data file?

Q-7. How do we tell a utility program what we want it to do?

Q-8. What is the term given to arranging data records in a predefined sequence or order?

Q-9. To sort a data file, what must you tell the sort program?

Q-10. What are report program generators used for?

PROGRAMMING LANGUAGES

Programmers must use a language that can be understood by the computer. Several methods can achieve human-computer communication. For example, let us assume the computer only understands French and the programmer speaks English. The question arises: How are we to communicate with the computer? One approach is for the programmer to code the instructions with the help of a translating dictionary before giving them to the processor. This would be fine so far as the computer is concerned; however, it would be very awkward for the programmer.

Another approach is a compromise between the programmer and computer. The programmer first writes instructions in a code that is easier to relate to English. This code is not the computer's language; therefore, the computer does not understand the orders. The programmer solves this problem by giving the computer another program, one that enables it to translate the instruction codes into its own language. This translation program, for example, would be equivalent to an English-to-French dictionary, leaving the translating job to be done by the computer.

The third and most desirable approach from an individual's standpoint is for the computer to accept and interpret instructions written in everyday English terms. Each of these approaches has its place in the evolution of programming languages and is used in computers today.

MACHINE LANGUAGES

With early computers, the programmer had to translate instructions into the machine language form that the computers understood. This language was a string of numbers that represented the instruction code and operand address(es).

In addition to remembering dozens of code numbers for the instructions in the computer's instruction set, the programmer also had to keep track of the storage locations of data and instructions. This process was very time consuming, quite expensive, and often resulted in errors. Correcting errors or making modifications to these programs was a very tedious process.

SYMBOLIC LANGUAGES

In the early 1950s, mnemonic instruction codes and symbolic addresses were developed. This improved the program preparation process by substituting letter symbols (mnemonic codes) for basic machine language instruction codes. Each computer has mnemonic codes, although the symbols vary among the different makes and models of computers. The computer still uses machine language in actual processing, but it translates the symbolic language into machine language equivalent. Symbolic languages have many advantages over machine language coding. Less time is required to write a program. Detail is reduced. Fewer errors are made. Errors which are made are easier to find, and programs are easier to modify.

PROCEDURE-ORIENTED LANGUAGES

The development of mnemonic techniques and macroinstructions led to the development of procedure-oriented languages. Macroinstructions allow the programmer to write a single instruction that is equivalent to a specified sequence of machine instructions. These procedure-oriented languages are oriented toward a specific class of processing problems. A class of similar problems is isolated, and a language is developed to process these types of applications. Several languages have been designed to process problems of a scientific-mathematical nature and others that emphasize file processing.

Procedure-oriented languages were developed to allow a programmer to work in a language that is close to English or mathematical notation. This improves overall efficiency and simplifies the communications process between the programmer and the computer. These languages have allowed us to be more concerned with the problems to be solved rather than with the details of computer operation. For example:

COBOL (COmmon Business Oriented Language) was developed for business applications. It uses statements of everyday English and is good for handling large data files.

FORTRAN (FORmula TRANslator) was developed for mathematical work. It is used by engineers, scientists, statisticians, and others where mathematical operations are most important.

BASIC (Beginner's All-Purpose Symbolic Instruction Code) was designed as a teaching language to help beginning programmers write programs. Therefore, it is a general-purpose, introductory language that is fairly easy to learn and to use. With the increase in the use of microcomputers, BASIC has regained popularity and is available on most microcomputer systems.

Other languages gaining in popularity are PASCAL and Ada. PASCAL is being used by many colleges and universities to teach programming because it is fairly easy to learn; yet is a more powerful language than BASIC. Although PASCAL is not yet a standardized language, it is still used rather extensively on microcomputers. It has greater programming capabilities on small computers than are possible with BASIC.

Ada's development was initiated by the U.S. Department of Defense (DOD). Ada is a modern general-purpose language designed with the professional programmer in mind and has many unique features to aid in the implementation of large scale applications and real-time systems. Because Ada is so strongly supported by the DOD and other advocates, it will become an important language like those previously mentioned. Its primary disadvantage relates to its size and complexity, which will require considerable adjustment on the part of most programmers.

The most familiar of the procedure-oriented languages are BASIC and FORTRAN for scientific or mathematical problems, and COBOL for file processing.

Programs written in procedure-oriented languages, unlike those in symbolic languages, may be used with a number of different computer makes and models. This feature greatly reduces reprogramming expenses when changing from one computer system to another. Other advantages to procedure-oriented languages are (1) they are easier to learn than symbolic languages; (2) they require less time to write; (3) they provide better documentation; and (4) they are easier to maintain. However, there are some disadvantages of procedure-oriented languages. They require more space in memory, and they process data at a slower rate than symbolic languages.

Q-11. With early computers, the programmer had to translate instructions into what type of language form?

Q-12. When were mnemonic instruction codes and symbolic addresses developed?

Q-13. What led to the development of procedure oriented languages?

Q-14. What computer language was developed for mathematical work?

Q-15. What are two disadvantages of procedure oriented languages?

PROGRAMMING

Programming is, simply, the process of planning the computer solution to a problem. Thus, by writing:

1. Take the reciprocal of the resistance of all resistors (expressed in ohms);

2. Sum the values obtained in step 1;

3. Take the reciprocal of the sum derived in step 2.

A generalized process or program for finding the total resistance of a parallel resistance circuit has now been derived.

To progress from this example to preparing a program for a computer is not difficult. However, one basic characteristic of the computer must be kept in mind. It cannot think. It can only follow certain commands, and these commands must be correctly expressed and must cover all possibilities. Thus, if a program is to be useful in a computer, it must be broken down into specifically defined operations or steps. Then the instructions, along with other data necessary for performing these operations or steps, must be communicated to the computer in the form of a language or code that is acceptable to the machine. In broad terms, the computer follows certain steps in executing a program. It must first read the instructions (sequentially unless otherwise programmed), and then in accordance with these instructions, it executes the following procedures:

1. Locates the parameters (constants) and such other data as may be necessary for problem solution

2. Transfers the parameters and data to the point of manipulation

3. Manipulates the parameters and data in accordance with certain rules of logic
4. Stores the results of such manipulations in a specific location
5. Provides the operator (user) with a useful output

Even in a program of elementary character such as the one above, this would involve breaking each of the steps down into a series of machine operations. Then these instructions, parameters, and the data necessary for problem solution must be translated into a language or code that the computer can accept.

Next, we'll provide an introduction to the problem solving concepts and flow charting necessary to develop a program.

OVERVIEW OF PROGRAMMING

Before learning to program in any language, it is helpful to establish some context for the productive part of the entire programming effort. This context comprises the understanding and agreement that there are four fundamental and discrete steps involved in solving a problem on a computer.

The four steps are as follows:

1. State, analyze, and define the problem.
2. Develop the program logic and prepare a program flowchart or decision table.
3. Code the program, prepare the code in machine readable form, prepare test data, and perform debug and test runs.
4. Complete the documentation and prepare operator procedures for implementation and production.

Figure 3-2 depicts the evolution of a program. Programming can be complicated, and advance preparation is required before you can actually start to write or code the program. The first two steps, problem understanding/definition and flowcharting, fall into the advance planning phase of programming. It is important at this point to develop correct habits and procedures, since this will prevent later difficulties in program preparation.

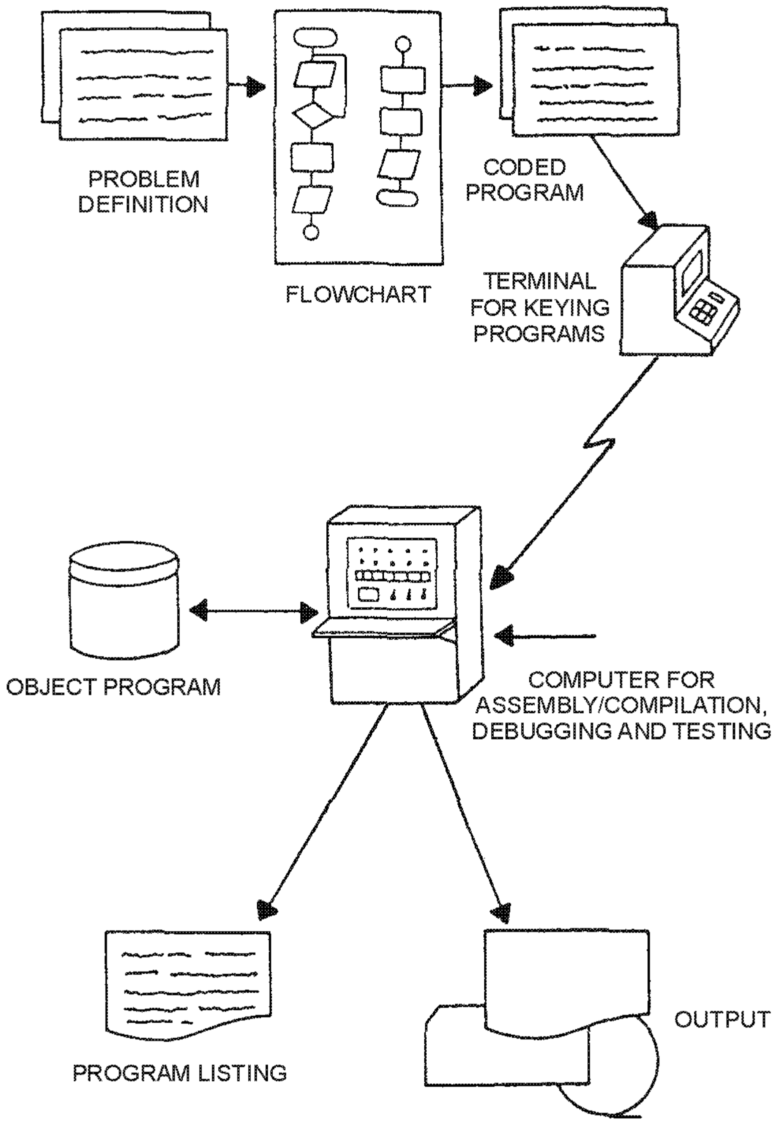

PROBLEM DEFINITION
FLOWCHART
CODED PROGRAM
TERMINAL FOR KEYING PROGRAMS
OBJECT PROGRAM
COMPUTER FOR ASSEMBLY/COMPILATION, DEBUGGING AND TESTING
PROGRAM LISTING
OUTPUT

Figure 3-2.—Evolution of a program.

Whether you are working with a systems analyst, a customer, or solving a problem of your own, it is extremely important that you have a thorough understanding of the problem.

Every aspect of the problem must be defined:

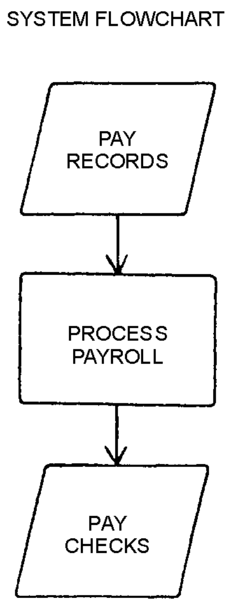
- What is the problem?
- What information (or data) is needed?
- Where and how will the information be obtained?
- What is the desired output?

Starting with only a portion of the information, or an incomplete definition, will result in having to constantly alter what has been done to accommodate the additional facts as they become available. It is easier and more efficient to begin programming after all of the necessary information is understood. Once you have a thorough understanding of the problem, the next step is flowcharting.

FLOWCHARTING

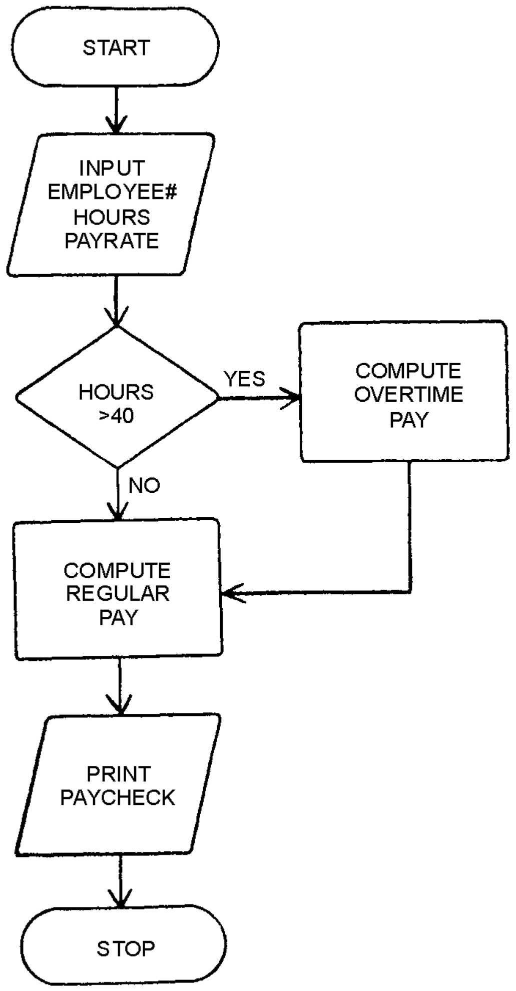
Flowcharting is one method of pictorially representing a procedural (step-by-step) solution to a problem before you actually start to write the computer instructions required to produce the desired results. Flowcharts use different shaped symbols connected by one-way arrows to represent operations, data flow, equipment, and so forth.

There are two types of flowcharts, system (data) flowcharts and programming flowcharts. A system (data) flowchart defines the major phases of the processing, as well as the various data media used. It shows the relationship of numerous jobs that make up an entire system. In the system (data) flowchart, an entire program run or phase is always represented by a single processing symbol, together with the input/output symbols showing the path of data through a problem solution. For example:



The second type of flowchart, and the one we will talk about in this section is the programming flowchart. It is constructed by the programmer to represent the sequence of operations the computer is to perform to solve a specific problem. It graphically describes what is to take place in the program. It displays specific operations and decisions, and their sequence within the program. For example:

PROGRAMMING FLOWCHART



Tools of Flowcharting

Next we will take a look at the tools used in flowcharting. These tools are the fundamental symbols, graphic symbols, flowcharting template, and the flowcharting worksheet.

FUNDAMENTAL SYMBOLS.—To construct a flowchart, you must know the symbols and their related meanings. They are standard for the military, as directed by *Department of the Navy Automated Data Systems Documentation Standards*, SECNAVINST 5233.

Symbols are used to represent functions. These fundamental functions are processing, decision, input/output, terminal, flow lines, and connector symbol. All flowcharts may be initially constructed using only these fundamental symbols as a rough outline to work from. Each symbol corresponds to one of the functions of a computer and specifies the instruction(s) to be performed by the computer. The contents of these symbols are called statements. Samples of these fundamental symbols, definitions, examples, and explanations of their uses are shown in figure 3-3.

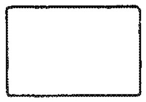
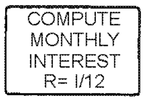
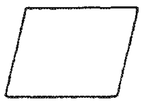
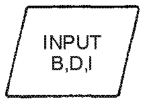
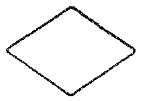
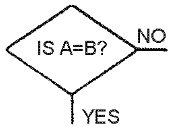
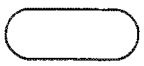
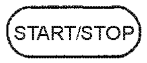
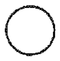
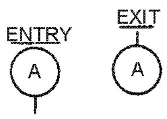
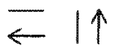
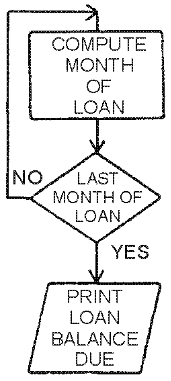
SYMBOL	DEFINITION	EXAMPLE	EXPLANATION
	**PROCESS SYMBOL** is used to represent general processing functions not represented by other symbols. It depicts the process of operations resulting in a change of value, form, or location of information.		Divide I by 12 assign value to R.
	**INPUT/OUTPUT SYMBOL** is used to represent any function of an I/O device. Making information available for processing is an Input function; recording processed information is an Output function.		Enter these values through the terminal, store in locations B, D, I.
	**DECISION SYMBOL** is used to depict a point in a program at which a branch to one of two or more alternate paths is possible.		If A is NOT equal to B, take NO branch. If A is equal to B, take YES branch.
	**TERMINAL, INTERRUPT SYMBOL** represents a terminal point in a flowchart, for example, start, stop, halt, delay, or interrupt.		START/STOP flow chart at this point.
	**CONNECTOR SYMBOL** represents a junction in a line of flow to another part of the flowchart. A common identifier, such as an alphabetic character, number, or mnemonic label, is placed within the exit and its associated entry.		This represents the EXIT point and the ENTRY point in a flowchart.
	**FLOWLINE SYMBOL** is used to represent flow direction by lines drawn between symbols. Normal direction of flow is left to right and top to bottom. If the direction of flow is other than normal, arrowheads are required at the point of entry.	 COMPUTE MONTH OF LOAN NO LAST MONTH OF LOAN YES PRINT LOAN BALANCE DUE	Initial processing is shown here. If the NO branch is taken, the processing block is performed again. If the YES branch is taken, the INPUT/OUTPUT operation is performed.

Figure 3-3.—Fundamental flowcharting symbols.

GRAPHIC SYMBOLS.—Within a flowchart, graphic symbols are used to specify arithmetic operations and relational conditions. The following are commonly-used arithmetic and relational symbols.

+	plus, add
-	minus, subtract
*	multiply
/	divide
±	plus or minus
=	equal to
>	greater than
<	less than
≥	greater than or equal to
≤	less than or equal to
≠	not equal
YES or Y	Yes
NO or N	No
TRUE or T	True
FALSE or F	False

FLOWCHARTING TEMPLATE.—To aid in drawing the flowcharting symbols, you may use a flowcharting template. Figure 3-4 shows a template containing the standard symbol cutouts. A template is usually made of plastic with the symbols cut out to allow tracing the outline.

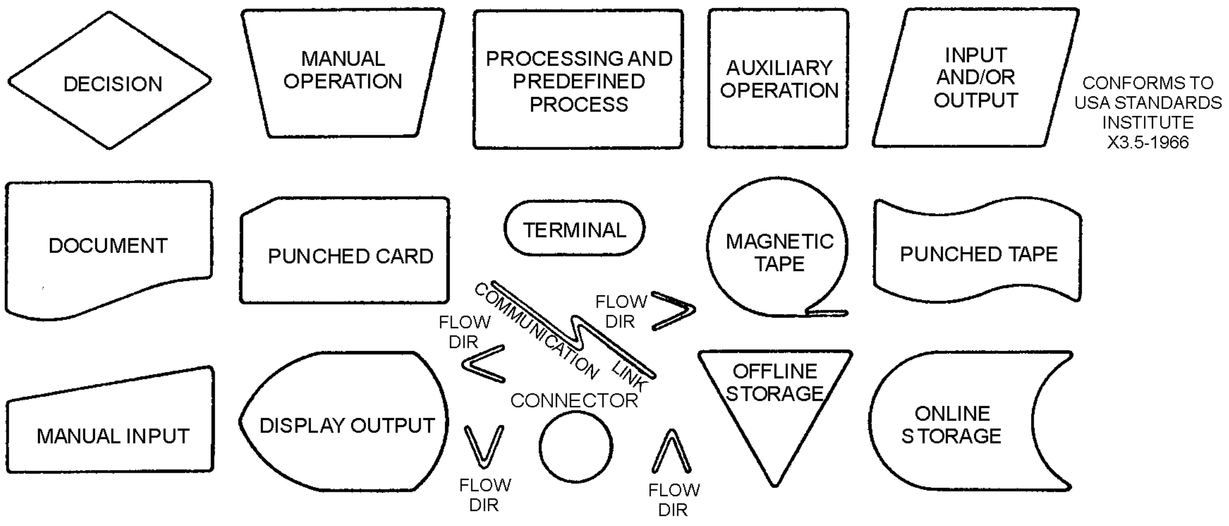


Figure 3-4.—Flowchart template.

FLOWCHART WORKSHEET.—The flowchart worksheet is a means of standardizing documentation. It provides space for drawing programming flowcharts and contains an area for identification of the job, including application, procedure, date, and page numbers (fig. 3-5). You may find it helpful when you develop flowcharts. If you don't have this form available, a plain piece of paper will do.

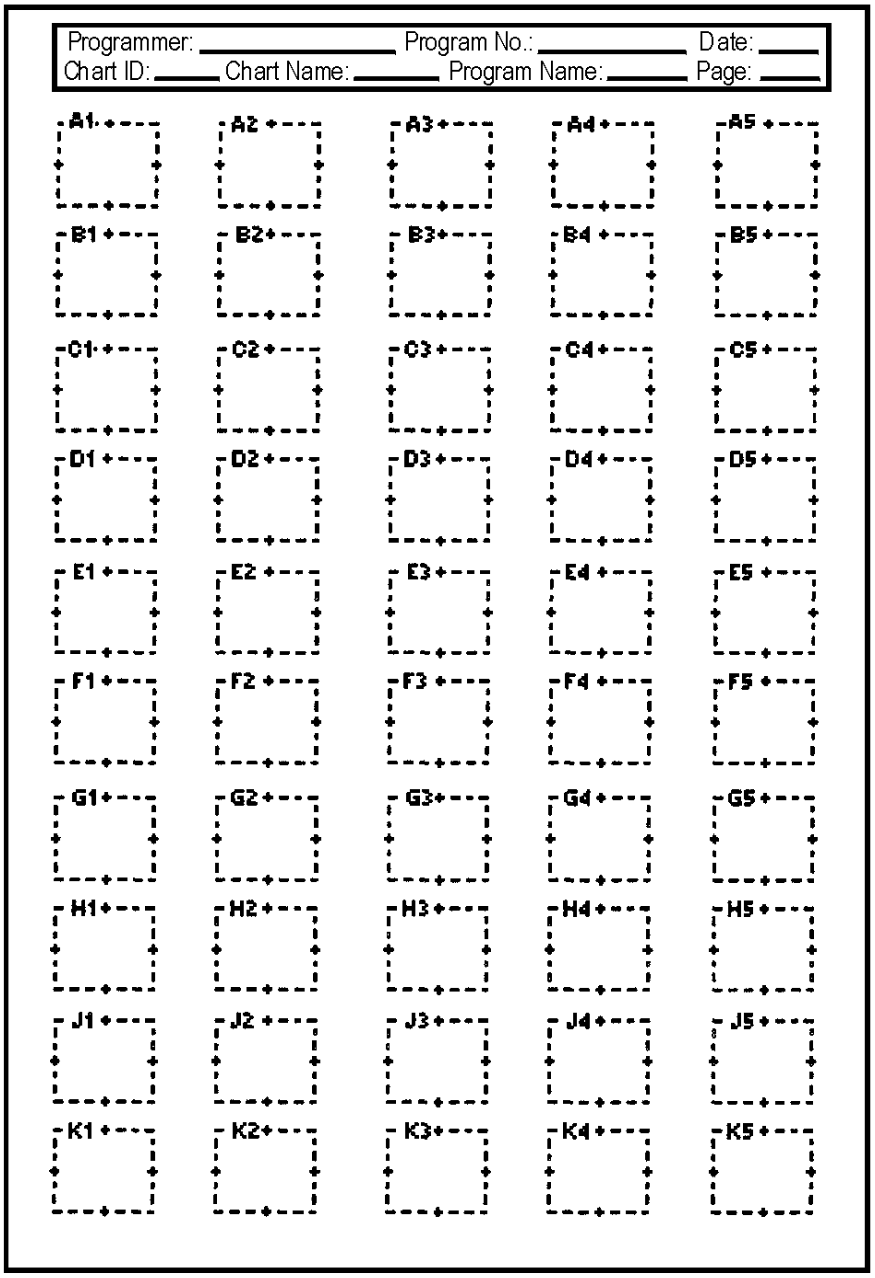


Figure 3-5.—Flowchart worksheet.

Constructing a Flowchart

There is no "best way" to construct a flowchart. There is no way to standardize problem solution. Flowcharting and programming techniques are often unique and conform to the individual's own methods or direction of problem solution.

This section will show an example of developing a programming flowchart. It is not the intent to say this is the best way; rather, it is one way to do it.

By following this text example you should grasp the idea of solving problems through flowchart construction. As you gain experience and familiarity with a computer system, these ideas will serve as a foundation.

To develop a flowchart, you must first know what problem you are to solve. It is then your job to study the problem definition and develop a flowchart to show the logic, steps, and sequence of steps the computer is to execute to solve the problem.

As an example, suppose you have taken a short-term second mortgage on a new home, and you want to determine what your real costs will be, the amount of interest, the amount to be applied to principal, and the final payment at the end of the 3-year loan period.

The first step is to be sure you understand the problem completely—What are the inputs and the outputs and what steps are needed to answer the questions? Even when you are specifying a problem of your own, you will find we don't usually think in small, detailed sequential steps. However, that is exactly how a computer operates, one step after another in a specified order. Therefore, it is necessary for you to think the problem solution through step by step. You might clarify the problem as shown by the Problem Definition in figure 3-6.

PROBLEM DEFINITION

MORTGAGE AMORTIZATION- This program is to determine the monthly amount of interest (A) and amount applied to the principal (P) of the mortgage giving the balance (B) at the end of a thirty-six month period.

INPUT: The monthly payment is to be entered as variable D, the beginning balance of the mortgage is to be entered as variable B, and the annual interest rate is to be entered as variable I. This input is to be entered into the system via the terminal.

OUTPUT: The end result is to be a listing displaying the amount applied to principal and interest and the current loan balance each month, with one final entry showing the final payment on the mortgage.

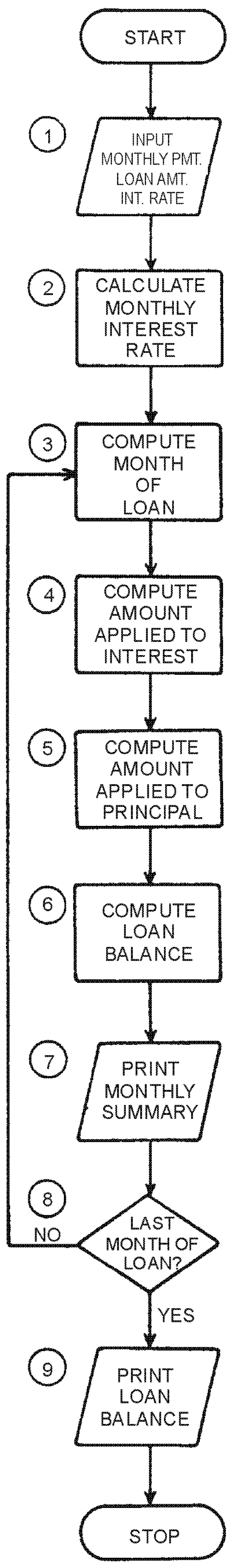

START
1
INPUT MONTHLY PMT. LOAN AMT. INT. RATE
2
CALCULATE MONTHLY INTEREST RATE
3
COMPUTE MONTH OF LOAN
4
COMPUTE AMOUNT APPLIED TO INTEREST
5
COMPUTE AMOUNT APPLIED TO PRINCIPAL
6
COMPUTE LOAN BALANCE
7
PRINT MONTHLY SUMMARY
8
LAST MONTH OF LOAN?
NO
YES
9
PRINT LOAN BALANCE
STOP

Figure 3-6.—Problem definition and programming flowchart.

After you have this level of narrative problem definition, you are ready to develop a flowchart showing the logic, steps, and sequence of steps you want the computer to execute to solve the problem. A programming flowchart of this problem is also shown in figure 3-6. Study both the problem definition and the flowchart to see their relationship and content.

You now have a plan of what you want the computer to do. The next step is to code a program that can be translated by a computer into a set of instructions it can execute. This step is called program coding.

PROGRAM CODING

It is important to remember program coding is not the first step of programming. Too often we have a tendency to start coding too soon. As we just discussed, a great deal of planning and preparation must be done before sitting down to code the computer instructions to solve a problem. For the example amortization problem (fig. 3-6), we have analyzed the specifications in terms of (1) the output desired; (2) the operations and procedures required to produce the output; and (3) the input data needed. In conjunction with this analysis, we have developed a programming flowchart that outlines the procedures for taking the input data and processing it into usable output. You are now ready to code the instructions that will control the computer during processing. This requires that you know a programming language.

All programming languages, FORTRAN, COBOL, BASIC and so on, are composed of instructions that enable the computer to process a particular application, or perform a particular function.

Instructions

The instruction is the fundamental element in program preparation. Like a sentence, an instruction consists of a subject and a predicate. However, the subject is usually not specifically mentioned; rather it is some implied part of the computer system directed to execute the command that is given. For example, the chief tells a sailor to "dump the trash." The sailor will interpret this instruction correctly even though the subject "you" is omitted. Similarly, if the computer is told to " ADD 1234," the control section may interpret this to mean that the arithmetic-logic section is to add the contents of address 1234 to the contents of the accumulator (a register in which the result of an operation is formed).

In addition to an implied subject, every computer instruction has an explicit predicate consisting of at least two parts. The first part is referred to as the command, or operation; it answers the question "what?" It tells the computer what operation it is to perform; i.e., read, print, input. Each machine has a limited number of built-in operations that it is capable of executing. An operation code is used to communicate the programmer's intent to the computer.

The second specific part of the predicate, known as the operand, names the object of the operation. In general, the operand answers the question "where?" Operands may indicate the following:

1. The location where data to be processed is found.
2. The location where the result of processing is to be stored.
3. The location where the next instruction to be executed is found. (When this type of operand is not specified, the instructions are executed in sequence.)

The number of operands and the structure or format of the instructions vary from one computer to another. However, the operation always comes first in the instruction and is followed by the operand(s). The programmer must prepare instructions according to the format required by the language and the computer to be used.

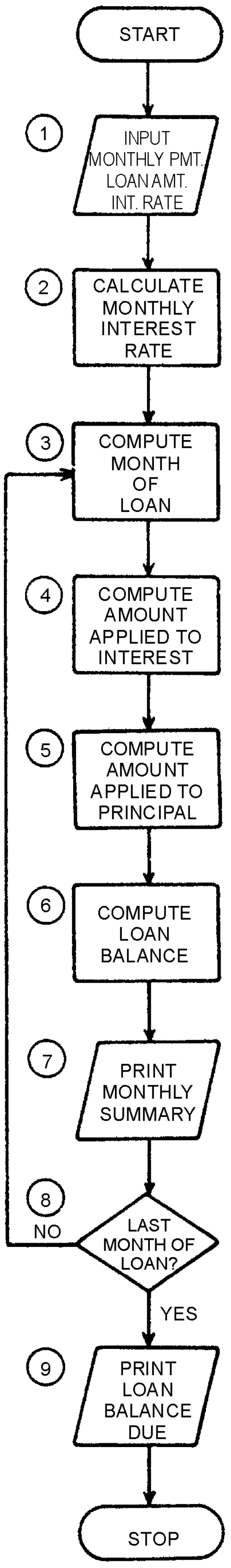
Instruction Set

The number of instructions in a computer's instruction set may range from less than 30 to more than 100. These instructions may be classified into categories by the action they perform such as input/output (I/O), data movement, arithmetic, logic, and transfer of control. Input/output instructions are used to communicate between I/O devices and the central processor. Data movement instructions are used for copying data from one storage location to another and for rearranging and changing of data elements in some prescribed manner.

Arithmetic instructions permit addition, subtraction, multiplication, and division. They are common in all digital computers. Logic instructions allow comparison between variables, or between variables and constants. Transfer of control instructions are of two types, conditional and unconditional. Conditional transfer of control instructions are used to branch or change the sequence of program control, depending on the outcome of the comparison. If the outcome of a comparison is true, control is transferred to a specific statement number. If it proves false, processing continues sequentially through the program. Unconditional transfer of control instructions are used to change the sequence of program control to a specified program statement regardless of any condition.

Coding a Program

Regardless of the language used, there are strict rules the programmer must adhere to with regard to punctuation and statement structure when coding any program. Using the programming flowchart introduced earlier, we have now added a program coded in BASIC to show the relationship of the flowchart to the actual coded instructions (fig. 3-7). Don't worry about complete understanding, just look at the instructions with the flowchart to get an idea of what coded instructions look like.


START
1
INPUT MONTHLY PMT. LOAN AMT. INT. RATE
2
CALCULATE MONTHLY INTEREST RATE
3
COMPUTE MONTH OF LOAN
4
COMPUTE AMOUNT APPLIED TO INTEREST
5
COMPUTE AMOUNT APPLIED TO PRINCIPAL
6
COMPUTE LOAN BALANCE
7
PRINT MONTHLY SUMMARY
8
LAST MONTH OF LOAN?
NO
YES
9
PRINT LOAN BALANCE DUE
STOP

```
10 PRINT "ENTER MONTHLY PAYMENT, ";
20 PRINT "LOAN AMOUNT, INTEREST RATE"
30 INPUT D,B,I
40 LET R= I/12
50 FOR M = 1 TO 36
60 LET A = B*R
70 LET P = D-A
80 LET B = B-P
90 PRINT M;D;B,P,A
100 NEXT M
110 PRINT "WITH ONE FINAL PAYMENT OF" ,B
120 END
```

Figure 3-7.—Programming flowchart and coded program.

You will have to have specific information about the computer you are to use. It will tell you how the language is implemented on that particular computer, in order to code a program. The computer manufacturers or software designer will provide these specifics in their user's manual. Get a copy of the user's manual and study it before you begin to code. The differences may seem minor to you, but they may prevent your program from running.

Once coding is completed, the program must be debugged and tested before implementation.

Debugging

Errors caused by faulty logic and coding mistakes are referred to as "bugs." Finding and correcting these mistakes and errors that prevent the program from running and producing correct output is called "debugging."

Rarely do complex programs run to completion on the first attempt. Often, time spent debugging and testing equals or exceeds the time spent in program coding. This is particularly true if insufficient time was spent on program definition and logic development. Some common mistakes which cause program bugs are mistakes in coding punctuation, incorrect operation codes, transposed characters, keying errors, and failure to provide a sequence of instructions (a program path) needed to process certain conditions.

To reduce the number of errors, you will want to carefully check the coding sheets before they are keyed into the computer. This process is known as "desk-checking" and should include an examination for program completeness. Typical input data should be manually traced through the program processing paths to identify possible errors. In effect, you will be attempting to play the role of the computer. After you have desk checked the program for accuracy, the program is ready to be assembled or compiled. Assembly and compiler programs prepare your program (source program) to be executed by the computer, and they have error diagnostic features which detect certain types of mistakes in your program. These mistakes must be corrected. Even when an error-free pass of the program through the assembly or compiler program is accomplished, this does not mean your program is perfected. However, it usually means the program is ready for testing.

Testing

Once a program reaches the testing stage, generally, it has proved it will run and produce output. The purpose of testing is to determine that all data can be processed correctly and that the output is correct. The testing process involves processing input test data that will produce known results. The test data should include: (1) typical data, which will test the commonly used program paths; (2) unusual but valid data, which will test the program paths used to process exceptions; (3) incorrect, incomplete, or inappropriate data, which will test the program's error routines. If the program does not pass these tests, more testing is required. You will have to examine the errors and review the coding to make the coding corrections needed. When the program passes these tests, it is ready for computer implementation. Before computer implementation takes place, documentation must be completed.

Documentation

Documentation is a continuous process, beginning with the problem definition. Documentation involves collecting, organizing, storing, and otherwise maintaining a complete record of the programs and other documents associated with the data processing system.

The Navy has established documentation standards to ensure completeness and uniformity for computer system information between commands and between civilian and Navy organizations. SECNAVINST 5233.1 establishes minimum documentation requirements.

A documentation package should include:

1. A definition of the problem. Why was the program written? What were the objectives? Who requested the program, and who approved it? These are the types of questions that should be answered.

2. A description of the system. The system environment (hardware, software, and organization) in which the program functions should be described (including systems flowcharts). General systems specifications outlining the scope of the problem, the form and type of input data to be used, and the form and type of output required should be clearly defined.

3. A description of the program. Programming flowcharts, program listings, program controls, test data, test results, and storage dumps — these and other documents that describe the program and give a historical record of problems and/or changes should be included.

4. Operator instructions. Items that should be included are computer switch settings, loading and unloading procedures, and starting, running, and termination procedures.

Implementation

After the documentation is complete, and the test output is correct, the program is ready for use. If a program is to replace a program in an existing system, it is generally wise to have a period of parallel processing; that is, the job application is processed both by the old program and by the new program. The purpose of this period is to verify processing accuracy and completeness.

Q-16. What is programming?

Q-17. In programming, how many steps are involved in solving a problem on a computer?

Q-18. What is required before you can actually start to write or code a program?

Q-19. In flowcharting, what method is used to represent different operations, data flow, equipment, and so forth?

Q-20. What type of flowchart is constructed by the programmer to represent the sequence of operations the computer is to perform to solve a specific problem?

Q-21. How many tools are used in flowcharting?

Q-22. Is there a "best way" to construct a flowchart?

Q-23. What controls the computer during processing?

Q-24. What is the fundamental element in program preparation?

Q-25. What type of instructions permit addition, subtraction, multiplication, and division?

Q-26. Where is specific information about the computer you are to use contained?

Q-27. How do we refer to errors caused by faulty logic and coding mistakes?

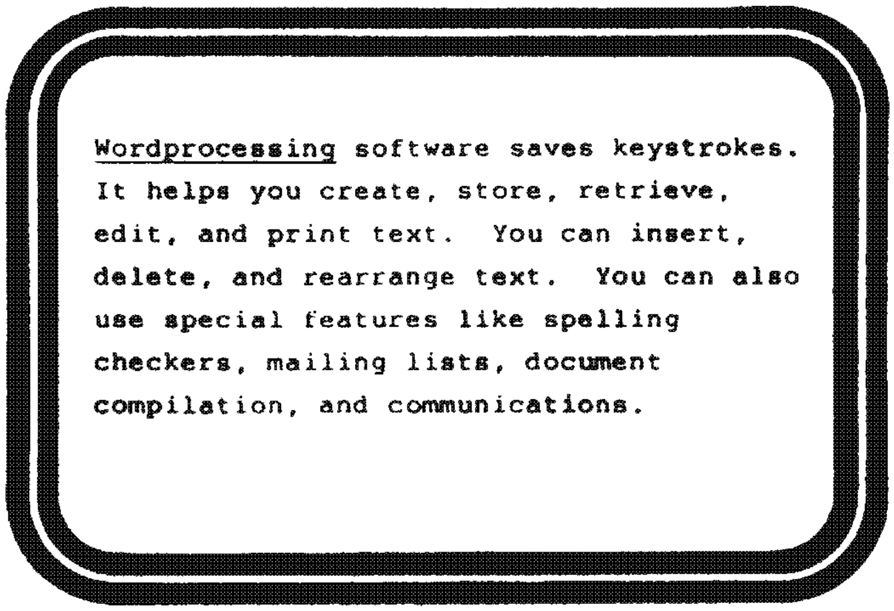
Q-28. What is the purpose of testing a program?

PACKAGED SOFTWARE

Fortunately you don't have to write a program for every problem to be solved. Instead, you can use packaged or off-the-shelf programs that are designed for specific classes of applications. Everyday more and more packaged software (software written by the manufacturer, a software house, or central design agency [CDA]) becomes available for general use. It may be up to you to set up and process a job within the specifications of a packaged program. Let's look at four classes of packaged software you may work with: word processing, data management, spreadsheets, and graphics.

WORD PROCESSING

You can use word processing software for any function that involves text: letters, memos, forms, reports, and so on. At a minimum, it includes routines for creating, editing, storing, retrieving, and printing text. Under the word processing software control, you generally enter the text on the keyboard and it is printed on a display screen as shown in figure 3-8. At that point, you may store it on disk or tape, print it on a printer, or change (edit) it. Using the edit functions you can add or delete words, characters, lines, sentences, or paragraphs. You can rearrange text; for example, move a paragraph or block of information to another place in the same document or even move it to a different document. Word processing is particularly useful for text documents that are repetitive or that require a lot of revisions. It saves a lot of rekeying.



```
Wordprocessing software saves keystrokes.
It helps you create, store, retrieve,
edit, and print text.  You can insert,
delete, and rearrange text.  You can also
use special features like spelling
checkers, mailing lists, document
compilation, and communications.
```

Figure 3-8.—Word processing example.

Other features and software often available with a word processing software package include: spelling checkers, mailing list programs, document compilation programs, and communications programs.

Spelling checker software helps find misspelled words but not misused words. It scans the text matching each word against a dictionary of words. If the word is not found in the dictionary, the system flags the word. You check it. If it is misspelled, you can correct it. You will still have to proofread the document to see that everything was keyed and that the words are used correctly.

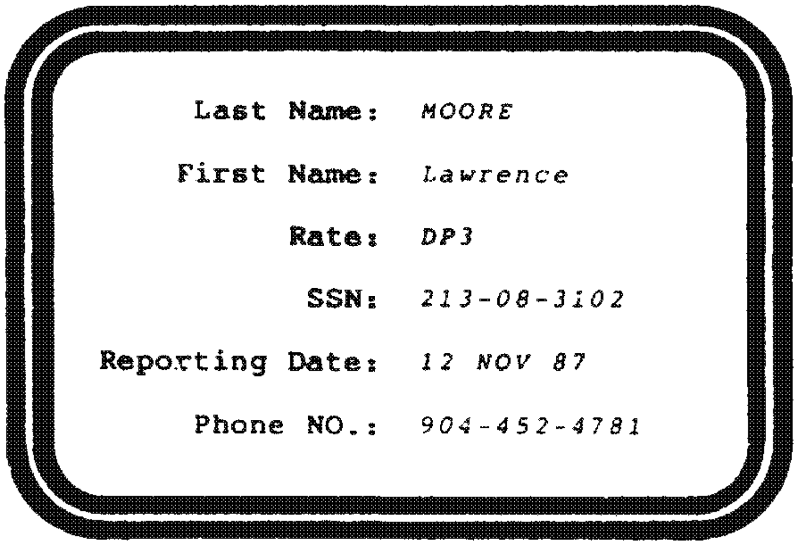
Mailing list programs are for maintaining name and address files. They often include a capability to individualize letters and reports by inserting names, words, or phrases to personalize them.

Document compilation programs are useful when you have standard paragraphs of information that you need to combine in different ways for various purposes. For example, you may be answering inquiries or putting together contracts or proposals. Once you select the standard paragraphs you want, you add variable information. This saves both keying and proofreading time.

Communications software and hardware enable you to transmit and receive text on your microcomputer. Many organizations use this capability for electronic mail. In a matter of minutes you can enter and transmit a memo to other commands or to personnel in other locations. You can transmit monthly reports, notices, or any documents prepared on the microcomputer.

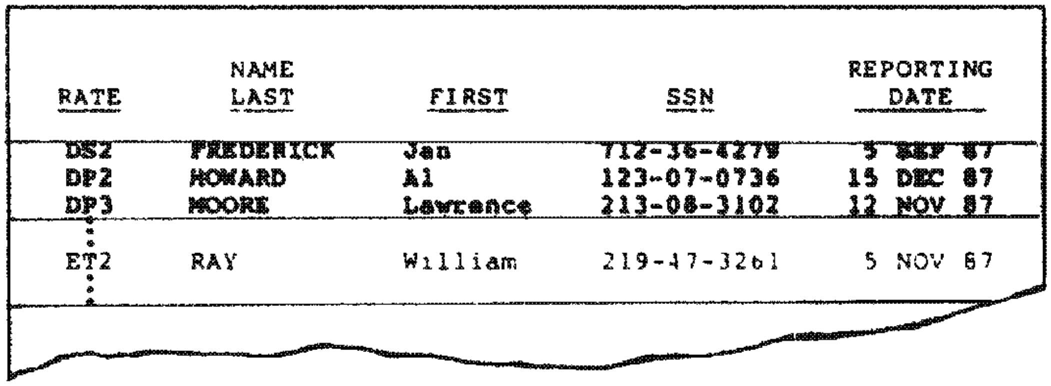
DATA MANAGEMENT

Data management software allows you to enter data and then retrieve it in a variety of ways. You define your data fields and set up a display screen with prompts. You enter the data records according to the prompts. Figure 3-9, view A, shows an example. The system writes the records on a disk or tape. Once you have a file keyed and stored, you can retrieve records by a field or several fields or by searching the records for specific data. For example, if you wanted a list of all personnel who reported aboard before January 1988, you could tell the system to search the file and print selected fields of all records that meet that condition. You tell the system what fields to print (that is name, rate, SSN, date reported) and where (what print positions) to print them. At the same time, you can specify in what order you want the records printed. For example, figure 3-9, view B, shows the records printed in alphabetical order by last name. The software also provides routines so you can easily add, delete, and change records.



Last Name:	*MOORE*
First Name:	*Lawrence*
Rate:	*DP3*
SSN:	*213-08-3102*
Reporting Date:	*12 NOV 87*
Phone NO.:	*904-452-4781*

Figure 3-9A.—Data management example. PROMPTS (IN BOLD) AND DATA (IN ITALICS).



```
         NAME                                            REPORTING
RATE     LAST         FIRST          SSN                   DATE

DS2      FREDERICK    Jan            712-36-4279          5 SEP 87
DP2      HOWARD       Al             123-07-0736         15 DEC 87
DP3      MOORE        Lawrence       213-08-3102         12 NOV 87
 :
ET2      RAY          William        219-47-3261          5 NOV 87
 :
```

Figure 3-9B.—Data management example. SAMPLE PRINTED REPORT (SORTED BY LAST NAME).

You can also generate reports by specifying what records to use, what fields to print, where to print the fields, and which data fields, if any, need to be combined. For example, your supply officer wants to know the value of the inventory. You can specify that the extended price is to be calculated by multiplying the item quantity by the unit price, and that the extended prices are to be totaled.

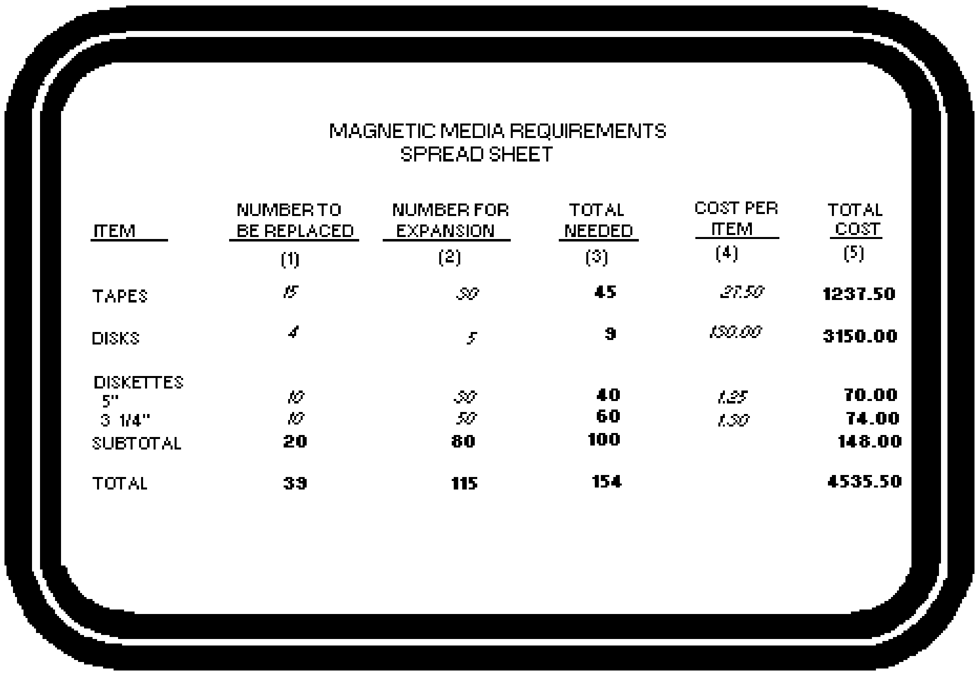
INVENTORY VALUE			
ITEM	QUANTITY	UNIT PRICE	EXTENDED PRICE
Swabs	47	1.65	77.55
Brooms	62	2.25	139.50
Foxtails	36	1.85	66.60
TOTAL			283.65

You can also specify the information to be used in report and column headings.

While the data management programs on micros are not as sophisticated as the data base management systems on mainframes and minis, they do provide an extremely useful capability in offices or aboard ship.

SPREADSHEETS

Spreadsheets are tables of rows and columns of numbers. Figure 3-10 shows an example. Spreadsheet processors allow you to set up a table of rows and columns and specify what calculations to perform on the columns. You enter values for the basic information into the appropriate rows and columns. Then the processor performs the calculations. In our example (fig. 3-10), we used a spreadsheet to project magnetic media costs. You enter the item descriptions, column headings, report title, and data for columns 1, 2, and 4, and the software calculates column 3 by adding columns 1 and 2. Then it multiplies column 3 times column 4 and puts the result in column 5. It also subtotals and totals the columns you specify; in this case, columns 1 through 3 and column 5.



MAGNETIC MEDIA REQUIREMENTS
SPREAD SHEET

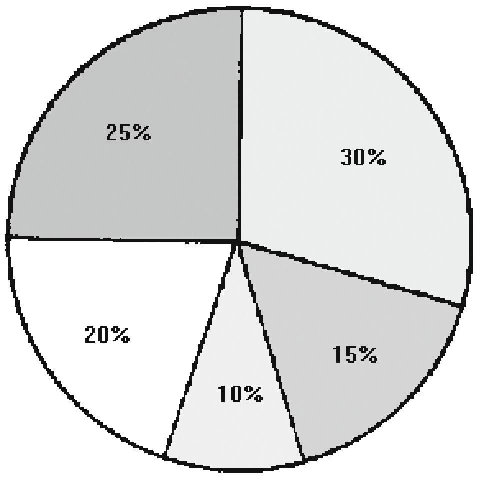
ITEM	NUMBER TO BE REPLACED (1)	NUMBER FOR EXPANSION (2)	TOTAL NEEDED (3)	COST PER ITEM (4)	TOTAL COST (5)
TAPES	*15*	*30*	**45**	*27.50*	**1237.50**
DISKS	*4*	*5*	**9**	*150.00*	**3150.00**
DISKETTES					
5"	*10*	*30*	**40**	*1.25*	**70.00**
3 1/4"	*10*	*50*	**60**	*1.30*	**74.00**
SUBTOTAL	**20**	**80**	**100**		**148.00**
TOTAL	**39**	**115**	**154**		**4535.50**

ITALICS - - DATA YOU ENTER
BOLD -- INFORMATION CALCULATED BY THE PROGRAM

Figure 3-10.—Spreadsheet example.

GRAPHICS

Graphics capability is available on many microcomputers. One use is to produce data displays, like bar charts, pie charts, and graphs. See figure 3-11, view A and view B. On some micros, you can do line drawings; on others you can create sophisticated engineering drawings. High resolution color graphics are also available for specialized applications.



A. PIE CHART

Figure 3-11A.—Graphics examples. PIE CHART.

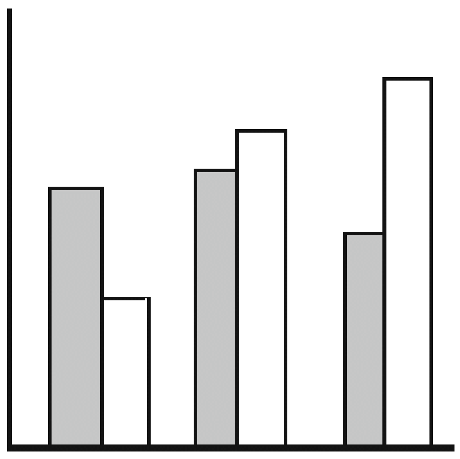


Figure 3-11B.—Graphics examples. BAR CHART.

You cannot use all printers for graphics output. They must be capable of producing graphics and also be compatible with the software. Some character printers can be used for limited graphics. Dot-matrix printers and plotters work well for graphics output. Laser and ink jet printers are also good for both text and graphics.

Q-29. What is packaged software?

Q-30. What are some of the other features and software available with a word processing software package?

Q-31. What software allows you to enter data and then retrieve it in a variety of ways?

Q-32. What are spreadsheets?

Q-33. Are all printers capable of handling graphics output?

SUMMARY

Congratulations you have just finished chapter 3. By now you should be convinced that anyone, with a little study, can understand digital computers. You probably thought when you first started this *NEETS* MODULE that it would get more difficult as your studies progressed. Our objective was to show you that the more you learn, the easier the material is to understand.

OPERATING SYSTEMS are a collection of many programs used by the computer to manage its own resources and operations and to perform commonly used functions like copy, print, and so on.

UTILITY PROGRAMS perform such tasks as sorting, merging, and transferring (copying) data from one input/output device to another: card to tape, tape to tape, tape to disk, and so on.

SORT-MERGE PROGRAMS arrange data records in a predefined sequence or order and are capable of combining two or more ordered files into one file.

REPORT PROGRAM GENERATORS are used to generate programs to print detail and summary reports of data files.

PROGRAMMING LANGUAGES are the means by which human-computer communication is achieved. They are used to write the instructions to tell the computer what to do to solve a given problem.

A **MACHINE LANGUAGE** uses a string of numbers that represent the instruction codes and operand addresses to tell the computer what to do.

SYMBOLIC LANGUAGES improved the program preparation process by substituting letter symbols (mnemonic codes) for basic machine language instruction codes.

A **PROCEDURE ORIENTED LANGUAGE** is a programming language oriented toward a specific class of processing problems. Examples are BASIC, COBOL, and FORTRAN.

PROGRAMMING is the process of planning and coding the computer instructions to solve a problem.

FLOWCHARTING is one method of pictorially representing a procedural (step-by-step) solution to a problem before you actually start to write the computer instructions required to produce the desired results.

PACKAGED SOFTWARE is designed for specific classes of applications. Examples are word processing, spreadsheets, data management, and graphics. These off-the-shelf programs are written by the manufacturer, a software house, or a central design agency.

ANSWERS TO QUESTIONS Q1. THROUGH Q33.

A-1. The operating system.

A-2. Single user/single tasking.

A-3. Multiuser/multitasking.

A-4. Because, to use applications software, it must be compatible with the operating system.

A-5. Some software comes in several versions so it can run under several different operating systems.

A-6. Utility programs.

A-7. By providing information about files, data fields, and the process to be used.

A-8. Sorting.

A-9. What data field or fields to sort on.

A-10. To generate programs to print detail and summary reports of data files.

A-11. Machine.

A-12. In the early 1950's.

A-13. The development of mnemonic techniques and macroinstructions.

A-14. FORTRAN.

A-15. They require more space in memory and they process data at a slower rate than symbolic languages.

A-16. The process of planning the solution to a problem.

A-17. Four.

A-18. Advance preparation.

A-19. Different shaped symbols.

A-20. A programming flowchart.

A-21. Four.

A-22. No, there isn't a way to standardize problem solution.

A-23. Coded instructions.

A-24. The instruction.

A-25. Arithmetic.

A-26. In the computer manufacturers or software designers user's manual.

A-27. Bugs.

A-28. To determine that all data can be processed correctly and that the output is correct.

A-29. Off-the-shelf programs designed for specific classes of applications.

A-30. Spelling checkers, mailing list programs, document compilation programs, and communications programs.

A-31. Data management.

A-32. They are tables of rows and columns of numbers.

A-33. No.